Zadanie č. 9
Tomáš Meravý Murárik

9. Vypísať obsah vstupu, pričom ak sa v ňom nachádza viacero rovnakých riadkov za sebou, vypíše sa vždy len jeden krát.

Bonusová úloha 7: Možnosť použitia súborov s veľkosťou nad 64 KB.
Bonusová úloha 9: Možnosť použitia viacerých súborov.
Bonusová úloha 11: Zmysluplné použitie reťazových inštrukcií (`movsb`, `cmpsb`).
Bonusová úloha 12: Okomentovaný program v anglickom jazyku.

03/23/2025
2. Ročník, 2. ak. rok, 2. semester, odbor Informatika

```
section    .data
filename1  db 'example.txt', 0; first file name
filename2  db 'double.txt', 0; second file name
;filename3 db '64kb.txt', 0; third file name
file_count   dq 2                    ; number of files in our table
buffer_size dq 69536
message    db 'closing', 10
fmt        db "%d", 10, 0; Format string with newline
newline    db 10

        ; Table of filename pointers:

filenames:
        dq  filename1
        dq  filename2
        ;dq filename3

        section .bss
        buffer  resb 69536; Buffer for file input
        current_buffer resb 512; Holds characters until newline/0
        prev_buffer    resb 512              ; Holds previous string
        num_buffer     resb 20               ; For debugging

        section .text
        global  main
        extern  printf

main:
        mov rbx, 0; index = 0

file_loop:
        mov rcx, [file_count]; rcx = file count
        cmp rbx, rcx
```

```
        jge all_files_done; if index >= file_count, we're done

        ;   Get the pointer to the filename from the table:
        mov rax, rbx
        shl rax, 3; Multiply index by 8 (size of pointer)
        lea rsi, [filenames + rax]; rsi points to the pointer in the table
        mov rdi, [rsi]; load the filename pointer into rdi

        ;   Open the file (sys_open)
        mov rax, 2; sys_open system call
        mov rsi, 0; Flags: read-only
        mov rdx, 0; Mode (not needed for read-only)
        syscall

        test rax, rax; Check if file was opened successfully
        js   next_file; If error, skip to next file

        mov rdi, rax; Store file descriptor in rdi

        ;   Initialize indices for file processing:
        mov r12, 0; index into buffer (buffer index)
        mov r14, 0; index in current_buffer
        mov r15, 0; index in prev_buffer

        ; ---- File Processing Loop ----

load_file:
        mov rax, 0; sys_read system call
        mov rsi, buffer; read into buffer
        mov rdx, buffer_size; buffer size
        syscall

        test rax, rax; if EOF (0) or error (<0)
        jle  close_current_file; then close the file

        mov r13, rax; r13 = number of bytes read
        jmp read_file_by_char

read_file_by_char:
        cmp r12, r13; if we've reached end of bytes read...
        jge close_current_file; close the file

        cmp byte [buffer + r12], 10; If newline encountered...
        je  cmp_strings

        ;   Append the current character from buffer into current_buffer:
        lea rdi, [current_buffer + r14]
        mov al, [buffer + r12]
        mov [rdi], al
```

```
        inc r12
        inc r14
        jmp read_file_by_char

cmp_strings:
        inc r12; move past the newline in buffer
        cmp r14, r15; compare current_buffer length vs. previous
        jne prepare_copy; if lengths differ, update prev_buffer

        lea rdi, [prev_buffer + r15 + 1]
        mov byte [rdi], 0; append null terminator to prev_buffer

        ;    Use rep cmpsb to compare strings efficiently:
        mov  rsi, current_buffer; pointer to current_buffer
        mov  rdi, prev_buffer; pointer to prev_buffer
        mov  rcx, r14; number of bytes to compare
        cld  ; clear direction flag
        repe cmpsb; compare bytes
        jnz  prepare_copy; if mismatch, copy new string

        ;   If strings are equal, reset current_buffer and continue:
        mov r14, 0
        jmp read_file_by_char

prepare_copy:
        ;   Copy current_buffer to prev_buffer using rep movsb:
        mov rsi, current_buffer; source pointer
        mov rdi, prev_buffer; destination pointer
        mov rcx, r14; number of bytes to copy
        mov r15, r14; save length for printing later
        cld ; ensure forward copying
        rep movsb; copy RCX bytes

        ;   Reset current_buffer by clearing it:
        mov rdi, current_buffer
        mov rcx, 512
        xor rax, rax
        rep stosb

        mov r14, 0; reset current_buffer index

        ;   Print the previous buffer:
        jmp print_message

print_message:
        mov rax, 1; sys_write for stdout
        mov rdi, 1
        mov rsi, prev_buffer; print prev_buffer
        mov rdx, r15; number of bytes to print
```

```
        syscall
        mov rax, 1; sys_write for newline
        mov rdi, 1
        mov rsi, newline
        mov rdx, 1
        syscall

        mov r14, 0; reset current_buffer index
        jmp read_file_by_char; continue processing current file

close_current_file:
        ;   Close the file (sys_close)
        mov rax, 3; sys_close
        syscall

        ; Move to next file:

next_file:
        inc rbx; next index in filenames table
        jmp file_loop

all_files_done:
        ;   All files processed; exit program
        mov rax, 60; sys_exit
        xor rdi, rdi
        syscall

        ;-----------------------------------------------

print_number:
        sub  rsp, 8; align stack
        mov  rdi, fmt; format string
        mov  rsi, rax; number to print
        xor  rax, rax
        call printf
        add  rsp, 8
        ret
```

Zhodnotenie:

Funkčnosť programu:

- **Program je funkčný** a splňuje všetky požiadavky zadania. Dokáže správne vypísať obsah vstupného súboru, pričom ak sa v ňom nachádza viacero rovnakých riadkov za sebou, vypíše každý takýto riadok len raz.
- **Prostredie**: Program bol vypracovaný v prostredí Linux a je napísaný v assembleri NASM.

Chovanie programu vzhľadom na vstupné údaje:

- **Obmedzenia**:
    - **Buffer size pre súbor**: Program používa buffer o veľkosti 69536 bajtov. Ak je vstupný súbor väčší ako tento buffer, môže dôjsť k buffer overflow.
    - **Buffer size pre riadok**: Program používa buffer o veľkosti 512 bajtov pre každý riadok. Ak je riadok dlhší ako 512 bajtov, môže dôjsť k buffer overflow.
- **Predpoklady správnej funkčnosti**:
    - Vstupný súbor sa musí zmestiť do buffera o veľkosti 69536 bajtov.
    - Každý riadok v súbore sa musí zmestiť do buffera o veľkosti 512 bajtov.

Chovanie programu vzhľadom na použité služby:

- Program používa systémové volania Linuxu (sys_open, sys_read, sys_write, sys_close) na prácu so súbormi a výstupom. Tieto služby sú použité správne a efektívne.

Možné vylepšenia:

- **Dynamická alokácia pamäte**: Namiesto statických bufferov by sa dala použiť dynamická alokácia pamäte, čo by umožnilo prácu s väčšími súbormi a dlhšími riadkami

Popis použitých algoritmov:

- **Hlavný algoritmus**:
    - Program prechádza každý súbor po znakoch a ukladá ich do `current_buffer`. Ak narazí na nový riadok, porovná `current_buffer` s `prev_buffer`. Ak sú rovnaké, ignoruje duplicitný riadok. Ak nie, skopíruje `current_buffer` do `prev_buffer` a vypíše ho.

Osobitosti riešenia:

- **Efektívne využitie reťazových inštrukcií**: Program efektívne využíva inštrukcie `movsb` a `cmpsb` na manipuláciu a porovnávanie reťazcov, čo zvyšuje jeho výkon.
- **Jednoduchosť a prehľadnosť**: Program je napísaný jednoducho a prehľadne, čo uľahčuje jeho pochopenie a úpravy